NIUF 404-92

North American ISDN Users' Forum Application Software Interface (ASI)

Part 3: Enhanced DOS/Protected Mode Shell Access Method (Version 1)

Approved: June 5, 1992

Updated: October 30, 1992

Application Software Interface Expert Working Group
ISDN Implementors' Workshop
North American ISDN Users' Forum

Revision History

June 1992	Baseline Approved Document (NIUF 404-92)
October 1992	Corrections and additions
March 1993	Editorial corrections

Abstract

This document describes the Application Software Interface (ASI) "Enhanced DOS/Protected Mode Shell Access Method". This access method is intended to operate on a DOS personal computer. This includes, but is not limited to, PC-DOS, MS-DOS and all compatible operating systems. Although it is a desired goal to have a common ASI for all operating systems, this specification will only apply to the DOS environment.

The following restrictions also apply:

- DOS Version 3.3 and above
- Microsoft Windows Version 3.0 and above

Keywords

application programming interface; API; Application Software Interface; ASI; access method; DOS; implementation agreement; Integrated Services Digital Network; ISDN

Notice of Disclaimer

Acknowledgments

NIST would like to acknowledge the NIUF Application Software Interface Expert Working Group, and especially the following individuals, for their valuable contributions to this document:

Kenneth A. Argo	ICL
Ron Bhanukitsiri	Digital Equipment Corp.
Cheng T. Chen	Teleos Communications, Inc.
Stephen Halpern	NYNEX Science and Technology
Frank Heath	Rockwell CMC
Chris Nix	IBM
Stephen Rogers	Electronic Data Systems
Chris Schmandt	MIT Media Lab
Ben Stoltz	Sun Microsystems, Inc.
Robert E. Toense	NIST
Adrian Viego	Bellcore
Wayne Yamamoto	Sun Microsystems, Inc.

1.0. Scope

This document describes the Application Software Interface (ASI) “Enhanced DOS/Protected Mode Shell Access Method”.

This access method is intended to operate on a DOS personal computer. This includes, but is not limited to, PC-DOS, MS-DOS and all compatible operating systems.

Although it is a desired goal to have a common ASI for all operating systems, this specification will only apply to the DOS environment.

The following restrictions also apply:

- DOS Version 3.3 and above
- Microsoft Windows Version 3.0 and above

Note: There are references to Microsoft Windows 3.1 in this document. These are for reference only at this time and do not affect system requirements.

2.0. Enhancements and Restrictions

The following enhancements have been made over the **DOS Access Method Version 1**:

- This access method allows DOS applications running in real mode and within a Virtual Machine (VM) to have access to the ASI.
- Facilities have been added to extend access to resident devices. These extensions utilize the “Multiplex Interrupt” (INT 2Fh) and will allow for an ASI Entity (AE), implemented as a device driver, to access the Address Resolution Device Driver (ARDD). These services should only be used for inter-device driver communications. An AE implemented as a Terminate but Stay Resident (TSR) application which utilizes these services will not be considered compliant with this specification.
- Facilities have been added to allow an AE or Program Entity (PE) to send messages to the ARDD directly (i.e., without using INT 21h facilities).
- This issue supports the capability for multiple AE and PE entities given the following restrictions:
    - Multiple AEs may exist ONLY if the ARDD provides support for such as described in the ARDD specification section.
    - An AE may be bound to one (1) PE at any time.
    - A PE may be bound to multiple AEs.
- Microsoft Windows 3.0 and above operating in “Enhanced Mode (80386 and above)” is supported. There is no provision at this time for “Standard Mode (80286)” or “Real Mode (8086)” operation. Standard Mode may be addressed in future versions of this document. Real Mode will not be addressed, as it will not be supported by Microsoft Windows beginning with version 3.1.
- Microsoft Windows Graphical User Interface (GUI) applications may access ARDD facilities through a Dynamic Link Library (DLL). The DLL must be placed in a fixed memory segment.

The following restrictions also apply:

- The AE must be a global entity. This means the AE code must be present within a device driver (loaded in “config.sys”) or a TSR application. In either case the AE must be loaded prior to any protected mode shell.

- The ARDD will hook the INT 2Fh vector. Once hooked the ARDD will process and use information that is placed over that channel by a protected mode shell. The ARDD will also process inter-device commands from AE(s). These services should only be used for inter-device driver communications. An AE implemented as a Terminate but Stay Resident (TSR) application which utilizes these services will not be considered compliant with this specification.

Note: Only the Microsoft Windows Enhanced Mode shell is currently supported. Future enhancements to this document will add support for the Microsoft Windows Standard Mode shell and the DOS-5 DOSSHELL. Other shells may be supported in future revisions of this document.

2.1. Overview

The ASI access method must support the transfer of information and thread of execution across the interface. To accomplish these functions and provide for implementation independence, the “Enhanced DOS/Protected Mode Shell Access Method” utilizes a number of thread transfer mechanisms commonly found in PC applications today.

The DOS environment supports only a single thread of execution and a single process. While this access method allows for a shell to be placed above the access method to alleviate some of these shortcomings, the main device interface does lay under the shell and thus is still within the DOS environment and must deal with its shortfalls.

In the DOS environment, the ASI must provide the basic mechanism to transfer the thread of execution across the interface. This mechanism must be available from either side of the interface. To accomplish this task the “Enhanced DOS/Protected Mode Shell Access Method” will utilize one or more of the following mechanisms in both the AE and PE:

- Far calls to addresses (similar to the “DOS Access Method”)
- Software interrupts (INT 60h - 66h)
- Multiplex Interrupt (INT 2Fh)

To resolve the parameters for these mechanisms, the “Enhanced DOS/Protected Mode Shell Access Method” has defined a dynamic linking mechanism referred to as the “Address Resolution Device Driver” (ARDD). The ARDD is defined in detail in the next section.

One difference between the bind procedure within the “Enhanced DOS/Protected Mode Shell Access Method” and that of its earlier relative, the “DOS Access Method” is the physical location of the remote entity's callback routine. In the “Enhanced DOS/Protected Mode Shell Access Method” this address resides within the code segment of the ARDD. In the “DOS Access Method”, the calling entity was passed an address that resided within the called entity's code segment. The callback functions still exist; they are just executed through the ARDD. Actual operation and use has not changed from that of the “DOS Access Method”. It has, however, been enhanced to support an extensive list of calling options.

ASI functionality requires that the ASI provide three basic services; management, control and data. The management and control plane services in the “Enhanced DOS/Protected Mode Shell Access Method” are implemented as a single ASI function. Multiple management/control (M/C) functions may be available to a PE if it has bound to multiple AEs. The number of such connections will depend on the level of support provided by the specific ARDD implementation (see the section titled “Overview of the Address Resolution Device Driver”, in this document, for further information). Data services are provided by means of a user plane (UP). The user plane service is defined as a separate ASI function. There may be more than one user plane function associated with an AE or PE. It should be mentioned however, that there may be a limitation on the actual number of call slots available to the entity. A call slot is a portion of the ARDD code that is responsible for passing packets to a bound entity. This number may be obtained by a call to the “Get Device Information” IOCTL (refer to the section titled “Get Device Information” in “IOCTL Facilities”, in this document).

The information transfer mechanism across the ASI will be specific to the access method function. For all management/control plane transfers the calling entity (AE or PE) will format a message. This message will be passed across the interface by reference (a message pointer). The called ASI access method management/control function will be responsible for transferring the contents of that message at the time of the call. As soon as the called entity returns to the calling entity the message pointer and its contents will be invalid.

The user plane function is similar to the management/control plane. Messages will be passed across the interface by reference (a message pointer). The called ASI access method user plane function will be responsible for transferring the contents of the message at the time of the call. As soon as the called entity returns to the calling entity the message pointer and its contents will be invalid.

3.0. Address Resolution Device Driver

The Address Resolution Device Driver (ARDD) is a generic address resolution utility defined for DOS. It is responsible for supplying a standard software level mapping layer between the ISDN vendor's hardware and software (AE) and an application that wishes to utilize ISDN services (PE).

3.1. Overview

The introduction of DOS Protected Mode (PM) shells has added a new level of complexity to device driver requirements. PM shells allow multiple applications to be loaded concurrently into a PC. Many PM shells support larger amounts of memory than is physically resident in the PC. This is accomplished by using part of the hard drive as a swap area for inactive portions of application code. This type of memory is called “virtual memory”. Lastly some of these shells create a multi-tasking environment for DOS applications. The ARDD must be able to exist in this environment.

Most of the ASI driver resides beneath the protected mode multi-tasking shell. This simplifies the services that must be supplied by an ARDD implementation. An ARDD must be able to monitor the status of any Virtual Machine (VM) with which it has an active connection.

Most protected mode shells provide some level of communication with a DOS device driver. Microsoft Windows Standard Mode (version 3.1 and above) and the DOS-5 DOSSHELL kernels supply similar services to a device driver about VM status. Microsoft Windows Enhanced Mode (version 3.0 and above) supplies no such services to a DOS device driver. It does, however, provide facilities to allow a device driver to force activation of a particular VM for servicing and callbacks. Microsoft has defined a new type of device driver in the enhanced mode; Virtual Device Driver (VxD). VxDs are required as a result of the multi-tasking environment created by the enhanced kernel. In enhanced mode, any number of applications may simultaneously request service from a single DOS device. This may be monitored and arbitrated by a VxD. VxDs also receive all indications about VM status. In this specification a VxD is used to receive these VM status messages and “broadcast” them to the ARDD for servicing (refer to the section titled “Virtual Device Driver Specifications”, in this document, for further details).

Note: Further information on Microsoft Windows Virtual Device Drivers is available in the “Microsoft Windows Device Development Kit”.

Even though this interface supports multiple entities, it is not necessary that this support be implemented. The developers of the ARDD should document the level of support provided by their ARDD. This includes the number of AE entry points. Services such as INT 2Fh facilities must be an integral part of all implementations, although the level of such support may vary. The capabilities of a specific driver will be available to an application via a call to the “Get Device Information” General IOCTL (refer to the section titled “Get Device Information”, in “IOCTL Facilities”, in this document).

Note: An “entry point” is a character device within the ARDD that is used to provide service to an AE or PE.

3.2. ARDD Specification

The ARDD is a character device driver which may supply different levels of support as defined in the following sections.

3.3. Basic ARDD Services

This section defines the basic facilities that must be supported by any ARDD implementation. An ARDD which provides only the level of support defined in this section may be used by real mode applications only. A task switcher such as Microsoft Windows may not act properly with an ARDD providing only basic support.

Note: An ARDD providing basic level support is compatible with the "DOS Access Method" as defined in part 2 of the ASI specification.

Note: In the following section an "entry point" refers to the character device within the ARDD that is used to provide service to an AE or PE.

1. The ARDD contains multiple devices logically linked together. The names shall be: "ARDDAEnn" and "ARDDPE00". The "nn" represents a numeric beginning with "00". The first name in the ARDD will be known as "ARDDAE00". Support for this device as well as "ARDDPE00" is mandatory. Additional devices, if supported, may be added by passing a command line argument during load time (refer to item 6 for information on command line parameters). There are no requirements as to the number of AE entry points an ARDD implementation must supply.
2. The device attribute word in the device header shall be coded such that the device is a character device that will support "open", "close", "IOCTL Read", "IOCTL Write" and "Generic IOCTL" calls. This value should be 0xC840.
3. The ARDD will allow one "open" per device (i.e., devices are non-sharable).
4. "Initialization", "Open", "Close", "IOCTL Read" and "Generic IOCTL" are the only commands that the ARDD will process.
5. An entry for the ARDD must be included in the "config.sys" file driver list prior to any AE that may utilize INT 2Fh facilities (refer to item 12 for INT 2Fh facilities supported by the ARDD).
6. Command syntax for loading the ARDD will be as follows:

Note: Parameters between []s are optional parameters.

```
DEVICE=<driver name> /AEM:xxh [/AE:n]
```

Example:

```
DEVICE=ASI_ARDD.SYS /AEM:57h /AE:2
```

The INT 2Fh handle for AE entry point(s) (AEM) will be 57h. The "h" following the "7" denotes hexadecimal (base 16). If the "h" is omitted, the value is assumed to be decimal (base 10).

Note: The "/AEM" is a required parameter. If this parameter is omitted, the ARDD will return an error and unload itself.

The ARDD will load and initialize two (2) AE entry points using the names "ARDDAE00" and "ARDDAE01".

Note: If no "/AE" parameter is included, the ARDD will assume the existence of one (1) AE entry point ("ARDDAE00"). This keeps the ARDD compatible with the "DOS Access Method" as described in part 2 of the ASI specification.

Note: If any parameters are invalid (i.e., "/AEM:A5" would be in error because an "A" is not a valid numeric for base 10), the ARDD will return an error and unload itself. Likewise a parameter greater than that supported by the ARDD will cause a load failure (i.e., /AE:nn greater than that supported by the ARDD device driver).

7. The ARDD should hook the INT 2Fh chain in order to block the loading of any shell which may cause erratic system operation (refer to Appendix I; Hooking the INT 2Fh Vector). Since an ARDD implemented with basic level support does not allow protected mode shell use, any attempt to load such shells should be blocked. Below are procedures which may be used to block the loading of some protected mode shells.

    a. When Microsoft Windows begins loading the Standard or Enhanced mode kernels it will broadcast an INT 2Fh with AX=1605h. In order to prevent the Windows kernel from loading the ARDD should place a non-zero value in the CX register and return (IRET). The ARDD should display an error message prior to the IRET since Microsoft Windows will not display any indication of the cause of the failure. (Refer to Appendix I; Preventing the Loading of Microsoft Windows.)

    Note: There is no broadcast by Microsoft Windows to terminate the loading of the Microsoft Windows Real Mode Kernel. However, real mode will not be supported by Microsoft Windows beginning with version 3.1.

    Note: There is no broadcast to terminate the loading of the Microsoft Task Switcher supplied by DOS 5.

8. During an AR-BIND procedure the ARDD should return a far pointer to a call slot within its own code segment. A call slot is a portion of the ARDD code that is responsible for passing packets to the entity with which the bind took place. Extreme care must be used when assigning call slots. This code must be either duplicated a number of times to allow for multiple call slots or code must be added to manage re-entrance.

9. Requests may be passed to the ARDD using one or more of the following procedures:

Note: Address Resolution (AR) messages are described in the section, ARDD Command Set Description, in this document.

    a. Format an AR message and pass it to the ARDD using Interrupt 21h(33), Function 44h(68), Subfunction 03h(03) (“Send Control Data to a Character Device”).

    b. Format an AR message and create a device driver request packet. Place far calls to the ARDD “strategy” and “interrupt” routines directly (refer to the section titled “Making Device Driver Calls Directly”, in this document). This address must have been previously acquired using the “Get Header Address” General IOCTL (refer to the section titled “Get Header Address” in “IOCTL Facilities”, in this document.)

    c. Format an AR message. Use INT 2Fh facilities to pass the information to the ARDD (refer to the section titled “INT 2Fh Facilities”, in this document, for further details) (refer to item 12 for INT 2Fh facilities supported by the ARDD).

Note: INT 2Fh facilities are available only to an AE implemented as a device driver. An AE implemented as a Terminate but Stay Resident (TSR) application which utilizes these services will not be considered compliant with this specification.

10. The ARDD must not modify the value returned in the AX register by the far entity except where specified.

11. Since DOS does not allow for a device driver to pass user defined return/status codes to an application, the ARDD should keep a local copy of the last return/status value to allow for an application to retrieve the REAL return/status code using the “Get Return/Status” IOCTL (refer to the section titled “Get Return/Status” in “IOCTL Facilities”; in this document.)

12. The ARDD will monitor the INT 2Fh chain for intra-device messages from AEs (refer to Appendix I; Hooking the INT 2Fh Vector). The values and parameters for these commands are documented in the section, INT 2Fh Facilities, in this document. Messages passed using INT 2Fh facilities will contain the AEMultiplex ID (AEM) in the AH register (refer to item 6 for information on command line parameters).

13. Since all M/C and UP calls go through the ARDD it becomes the responsibility of the ARDD to call the far entity with the buffer's address. Information may be passed to a bound entity via one of the following methods (refer to device functional flowchart #1):

    a. If the entity registered with a callback address (Callback_Type=0), then information will be passed on the STACK by a call to the entity's callback function.

       The STACK frame for ARDD or Management/Control callbacks will be formatted in the following manner (refer to AR-BIND_Mc request messages for more information on bind procedures and their parameters):

```
PUSH         WORD     MsgSize             ;  Bytes in message
PUSH         WORD     seg MsgPtr          ;  Segment of message
PUSH         WORD     offset MsgPtr       ;  Offset of message
```

       The STACK frame for a UP callback will be formatted in the following manner (refer to AR-BIND_Up request messages for more information on bind procedures and their parameters):

```
PUSH         WORD     DataSize            ;  Bytes in data buffer
PUSH         WORD     seg DataPtr         ;  Segment of actual buffer
PUSH         WORD     offset DataPtr      ;  Offset of actual buffer
PUSH         WORD     CtlSize             ;  Bytes in control message
PUSH         WORD     seg CtlPtr          ;  Segment of control message
PUSH         WORD     offset CtlPtr       ;  Offset of control message
```

    b. If an entity registered with a software interrupt vector (Callback_Type=1), then the ARDD will execute a software interrupt to the vector supplied in the bind with DS:SI pointing to the segment and offset of a buffer frame. If the software vector is 2Fh, the ARDD will place the INT2F_AX value, passed during the bind, in AX prior to execution of the interrupt.

       The buffer frame for ARDD or Management/Control messages will be formatted in the following manner (refer to AR-BIND_Mc request messages for more information on bind procedures and their parameters):

```
             WORD     offset MsgPtr       ; Offset of message
             WORD     seg MsgPtr          ; Segment of message
             WORD     MsgSize             ; Bytes in message
```

       The buffer frame for a UP message will be formatted in the following manner (refer to AR-BIND_Up request messages for more information on bind procedures and their parameters):

```
             WORD     offset CtlPtr       ; Offset of control message
             WORD     seg CtlPtr          ; Segment of control message
             WORD     CtlSize             ; Bytes in control message
             WORD     offset DataPtr      ; Offset of actual buffer
             WORD     seg DataPtr         ; Segment of actual buffer
             WORD     DataSize            ; Bytes in data buffer
```

    c. Once the ARDD has delivered the message to the proper entity it should return to the original caller with AX and the carry flag unmodified.

14. The ARDD will respond to the Generic IOCTL, Category=0Ah (ARDD), minor code=80h (“Get Device Information”) with a major value of 0x20 and minor value of 00h. All other information returned must be filled in by the ARDD (refer to the section titled “Get Device Information” in “IOCTL Facilities”, in this document).

15. The ARDD will respond to the Generic IOCTL, Category=0Ah (ARDD), minor code=81h (“Get Return/ Status”) with the value last returned in the “status” field (bits 0-7) of the DOS device driver request header (refer to the section titled “Get Return/Status” in “IOCTL Facilities”, in this document.)

16. The ARDD may receive a "close" to an access point at any time. If a "close" is received, the ARDD must check for the existence of any active binds. If active binds exist, the ARDD must send an AR-UNBIND_ALL indication message to each entity bound with the closed entity. The ARDD must also block any further calls to the closed entity by returning a "Not Bound" error to the caller with the carry flag set. If the ARDD has any active bind registrations to the closed entry point, then the registration flag should also be cleared. The ARDD is responsible for closing all active User Plane callback slots assigned to the closed entity. The ARDD will also reset the Mc callback as defined below:

    a. If the remaining entity is an AE, then the ARDD will reset the entity to a status of "registered". An AE will not have to re-register following a PE disconnect.

    b. If the remaining entity is a PE, then the ARDD will clear all register/bind status and will return the entity to an idle state (unbound/unregistered). A PE must re-register following an AE disconnect.

3.4. Microsoft Windows Standard Mode Support

This sections defines facilities that must be supported by the ARDD to ensure compatibility with Microsoft Windows operating in Standard Mode. These services are in addition to the services defined in the section, ARDD Basic Service, in this document.

Microsoft Windows operating in Standard Mode is not supported at this time.

3.5. Microsoft Windows Enhanced Mode Support

This sections defines facilities that must be supported by the ARDD to ensure compatibility with Microsoft Windows operating in Enhanced Mode. These services are in addition to the services defined in the section "ARDD Basic Service" in this document.

Note: Parameters between "[]" are optional parameters.

1. Command syntax for loading the ARDD will be as follows:

   ```
   DEVICE=<driver name> /AEM:xxh /PEM:xxh [/AE:n]
   ```

   Example:

   ```
   DEVICE=ASI_ARDD.SYS /AEM:57h /PEM:5Fh /AE:2
   ```

   The INT 2Fh handle for AE entry point(s) (AEM) will be 57h. The "h" following the "7" denotes hexa-decimal (base 16). If the "h" is omitted the value is assumed to be decimal (base 10).

   The INT 2Fh handle for PE entry point(s) (PEM) will be 5Fh. The "h" following the "F" denotes hexa-decimal (base 16). If the "h" is omitted the value is assumed to be decimal (base 10).

   Note: "/AEM" and "/PEM" are required parameters. If either of these parameters are omitted, then the ARDD will return an error and unload itself.

   The ARDD will load and initialize two (2) AE entry points using the names "ARDDAE00" and "ARD-DAE01".

   Note: If no "/AE" parameter is included, the ARDD will assume the existence of one (1) AE entry point ("ARDDAE00"). This keeps the ARDD compatible with the "DOS Access Method" as described in part 2.

   Note: If any parameters are invalid (i.e., "/AEM:A5" would be in error because an "A" is not a valid numeric for base 10), then the ARDD will return an error and unload itself. Likewise a parameter greater than that supported by the ARDD will cause a load failure (i.e., /AE:nn greater than that supported by the ARDD device driver).

2. The ARDD must keep a map of Virtual Machine IDs (VMIDs) which are ASI active. A VM that has “opened” the ARDD is considered ASI active. Once the VM has “closed” the ARDD it is no longer considered ASI active and may be removed from the map.

3. The ARDD will hook the “Multiplex Interrupt” (INT 2Fh) in order to allow the ARDD to receive, among other things, messages from the Microsoft Windows Kernel (refer to “Appendix I, Hooking the INT 2Fh Vector” and “INT 2Fh Facilities”, for further details). These messages include:

    a. “Enhanced Windows and 286 DOS Extender Initialization” (AX=1605h). This call should be used to block the loading of the 286 DOS Extender. The ARDD may also create instance data for PE entry point access at this time. In addition the ARDD will cause the VxD to load and pass the PEM value to the VxD (refer to the section titled “Virtual Device (VxD) Specification”, in this document for further details)

    b. “Enhanced Windows Initialization Complete” (AX=1608h). This informs the ARDD that Windows Enhanced Mode has completed initialization and is running.

    c. “Enhanced Windows Begin Exit” (AX=1609h). This is broadcast at the beginning of a normal Windows exit procedure.

    d. “Enhanced Windows and 286 DOS Extender Exit” (AX=1606h). This is broadcast prior to Microsoft Windows returning to DOS.

4. Once Microsoft Windows is running, the ARDD must use INT 2Fh facilities provided by Microsoft Windows to manage PEs that access the ARDD's services (refer to the section titled “INT 2Fh Facilities”, in this document, for further details).

    Commonly used functions include:

    a. “Get Current Virtual Machine ID” (AX=1683h). This service may be used by the ARDD to acquire the currently running VMID.

    b. “Switch VMs and Callback” (AX=1685h). If the above function does not return the VMID with which the ARDD wishes to communicate, then this facility may be used to force the VMID to become active. Once Windows has activated the requested VMID, a far call will be made to the callback address supplied in the request by the ARDD.

    c. “Begin Critical Section” (AX=1681h). This may be used to lock a VM in memory while the ARDD delivers the message. This function should be used with leniency since no other task will be allowed to run until the critical section has completed.

    d. “End Critical Section” (AX=1682h). This function is used to signal the end of a critical section. This call must always be made following a call to “Begin Critical Section”.

Note: If Microsoft Windows is running, the ARDD must verify that the proper VM is available and in memory.

5. The ARDD will monitor the INT 2Fh chain for any “Broadcast” (AL=10h) from a VxD. These messages will contain the PEM id in the AH register, as specified in item 1 above, and a VMID in the BX register. The CX register will contain a value which represents the new state of the VM. If the VMID represents a VM which is ASI active the ARDD should perform one of the following:

    a. If the value in CX represents VM_NOT_EXECUTEABLE (30h), then the ARDD should “close” the access point. Procedures for closing an access point are described in item 16 of the section titled “Basic ARDD Services”, in this document.

    b. If the value in CX represents VM_SUSPEND, then the ARDD should begin interception of all messages, on all bound planes to the VM. The ARDD will return “Interface unavailable” to the caller with the carry flag set. The ARDD will also set a “lost_message” flag.

c. If the value in CX represents VM_RESUME (20h) the ARDD should, if the "lost_message" flag was set (as specified in to step b above), call the AR_Callback function of the VM with an "AR-LOST_MESSAGE". The ARDD will also discontinue the interception of messages to the VM.

6. The ARDD is responsible for the management of each VM instance of a PE entry point. Microsoft Windows Enhanced Kernel supplies facilities to assist in this function (refer to the section titled "Enhanced Windows and 286 Extender Init", in this document.)

3.5.1. Virtual Device (VxD) Specification

This section defines the protected mode portion of the ARDD that is responsible for broadcasting messages to the real mode ARDD about the status of VMs within Microsoft Windows. Services provided in this section may only be utilized by an ARDD which has "Enhanced Windows" support as defined in the section titled "ARDD Enhanced Windows Support", in this document.

A Microsoft Windows Virtual device driver (VxD) will have the following characteristics:

1. A VxD will be responsible for broadcasting messages to the ARDD using the INT 2Fh "BROADCAST" message (refer to the section titled "INT 2Fh Facilities", in this document for a list of valid message values). These messages should be broadcast using the PEM value from the ARDD load line in "config.sys" (refer to item 1 in the section titled "ARDD Enhanced Windows Support", in this document.)
2. The VxD shall be named "ASIARDD.386".
3. The VxD will be loaded by the ARDD during Windows Startup. The VxD shall receive a parameter from the ARDD which shall contain the PEM value. The VxD will place this value in AH prior to any status broadcasts.

3.6. DOS Task Switcher Support

This section defines facilities that must be supported by the ARDD to ensure compatibility with the Microsoft DOS Task Switcher. These services are in addition to the services defined in the section titled "ARDD Basic Service", in this document.

The Microsoft Task Switcher is not supported at this time.

3.7. Other Protected Mode Shells

For further study.

4.0. ARDD Message Format

Messages are sent to the ARDD using one of the methods outlined in "Basic ARDD Services, item 13" in the section titled "ARDD Specifications", in this document.

ARDD messages are comprised of a fixed length part (header) and a variable length part (body). The header contains one (1) byte which is composed of a command value (bits 0-5) and a flag field (bits 6-7) (refer to the section titled "ARDD Command Format", in this document, for further details). Most ARDD messages require additional bytes in order to complete their task. The actual number of additional bytes is dependent upon the ARDD command.

4.1. ARDD Command Format

The ARDD command byte is a combination command value and flag field as described as follows:

Bit(s)	Description		
0-5	Command Value	//	see command set below
6		//	reserved
7	Buffer control	//	if set during a bind, messages passed over that plane will contain a pointer which must remain valid until a response is received (this will be utilized in future enhancements to this document).

4.2. ARDD Command Set

Valid ARDD commands are:

AR-BIND_Mc confirmation	0x01	// Mc=Management/control
AR-BIND_Mc request	0x02	
AR-BIND_Up confirmation	0x03	// Up=User plane
AR-BIND_Up request	0x04	
AR-UNBIND_Mc indication	0x05	
AR-UNBIND_Mc request	0x06	
AR-UNBIND_Up indication	0x07	
AR-UNBIND_Up request	0x08	
AR-UNBIND_ALL indication	0x09	
AR-LOST_MESSAGE indication	0x10	

Note: There are no “response” primitives in the “Enhanced DOS/Protected Mode Shell Access Method”. Future enhancements to this document may contain “response” primitives.

5.0. ARDD Command Set Description

This section contains descriptions of the messages associated with each ARDD command. When reading this chapter, the following criteria apply:

- Each section title contains the Command Name in bold letters followed by an indicator which denotes the direction of the command (request, confirmation, indication or response).

Note: There are no “response” primitives in the “Enhanced DOS/Protected Mode Shell Access Method”. The explanation of the primitive was listed as a general definition for the ASI. Future enhancements to this document may contain “response” primitives.

- The title ends with the actual hex value of the command (bits 0-5).
- Following each title is a brief description of the command and a listing of the required parameters.
- Return values will be placed in the AX register unless otherwise specified.

5.1. ARDD Request Messages

"Request" messages are formed by the AE or PE and sent to the ARDD using one of the supported transfer mechanisms. "Request" messages are used by the AE or PE to send command requests to the ARDD.

5.1.1. AR-BIND_Mc request [0x02]

Instructs the ARDD to bind its management plane with that of a specific entity.

The bind instruction may use one of the following definitions. The actual use depends on the callback facility that a particular entity wishes to use for communications. Optional_1 refers to an AR callback likewise Optional_2 refers to the Mc callback.

Byte	Description		
2-3	Bind_Id	//	see description below
4-5	unassigned		
6-9	Optional_1	//	see description below
10-13	Optional_2	//	see description below
14	Callback_Type	//	see description below
15-18	Optional_Extension_1	//	see description below
19-22	Optional_Extension_2	//	see description below

The "Bind_Id" field has the following meaning:

- An AE calling the bind procedure uses this field to identify itself from other AEs in the system. This value is supplied to the AE at configure time and must be unique. The ARDD will return an error if multiple AEs attempt to register using the same "Bind_Id".

  Note: In order to remain compatible with the DOS Access Method, a "Bind_Id" value of zero (0) may be used by any number of AE(s) without causing an error.

- The PE uses this field to request a bind to a specific AE. If this value is "0", the ARDD will complete the bind to the first (1st) available AE. If an AE is not registered, the ARDD will register the PE.

  Note: The PE may have many bind requests outstanding. The total number of binds and bind requests may not exceed the number of AE entry points supported by the ARDD. If the PE attempts to make a bind request which lies outside the above parameters, the ARDD should return a 0x00A4 (Active Register/Bind) error.

Note: Any number of PE bind requests may carry a "Bind_Id" value of zero (0).

Callback_Type is a nibble-encoded field defined as follows:

bits 0-3	AR Callback type
bits 4-7	Mc Callback type

Refer to the following sections to define callback facilities. Byte values represent relative offset within the Optional_1 or Optional_2 field.

if Callback_Type=0 or is not present:

1-2	Offset Address
3-4	Segment Address

if Callback_Type=1:

1	Software IRQ	//	IRQ to use for callback
2	reserved		
3-4	INT2F_AX	//	AX for INT 2Fh callback if software IRQ=2Fh

Note: Optional_Extension_1 and Optional_Extension_2 are reserved for future enhancements.

Return:

Note: If the carry flag is set an error may have occurred. If Interrupt 21h(33), Function 44h(68), Subfunction 03h(03) ("Send Control Data to a Character Device") was used to send this message then the value in AX will contain a DOS mapped error value. This value may not represent the ASI error value and should be ignored. To retrieve the actual error/status value the AE or PE must use the "Get Return/Status" IOCTL (refer to the section "Get Return/Status", in "IOCTL", in this document). The value returned in the buffer by the IOCTL may be one of the following:

0x0000	OK	//	operation successful
0x0081	Message length error	//	invalid message length
0x0082	Invalid message pointer	//	invalid message pointer
0x0083	Out of Memory	//	insufficient memory to complete request
0x00A1	Registered	//	other entity not present
0x00A2	Invalid_Reference	//	invalid Reference_Id
0x00A3	Reference_not_unique	//	re-registered id
0x00A4	Active Register/Bind	//	id already bound
0x00A6	Missing or Illegal element	//	one or more field(s) are missing or contain invalid element values

5.1.2. AR-BIND_Up request [0x04]

Instructs the ARDD to bind its user plane with that of a specific entity.

The bind instruction may use one of the following definitions. The actual use depends on the callback facility a particular entity wishes to use for communications.

Byte	Description		
2-5	Reference_Id	//	AEI-PEI pair
6-9	Optional	//	see description below
10	Callback_Type	//	see description below
11-14	Optional_Extension	//	see description below

Note: The Reference_Id contains a copy of the four octets (AEI/PEI), 2 through 5, as defined in Part 1, Commands chapter, Message Format section.

Callback_Type is a nibble-encoded field defined as follows:

bits 0-3	AR Callback type
bits 4-7	Mc Callback type

Refer to the following sections to define callback facilities. Byte values represent relative offset within the Optional field.

if Callback_Type=0 or is not present:

1-2	Offset Address
3-4	Segment Address

if Callback_Type=1:

1	Software IRQ	//	IRQ to use for callback
2	reserved		
3-4	INT2F_AX	//	AX for INT 2Fh callback if software IRQ=2Fh

Note: The Optional_Extension field is reserved for future enhancements.

Return:

Note: If the carry flag is set, an error may have occurred. If Interrupt 21h(33), Function 44h(68), Subfunction 03h(03) ("Send Control Data to a Character Device") was used to send this message, then the value in AX will contain a DOS mapped error value. This value may not represent the ASI error value and should be ignored. To retrieve the actual error/status value the AE or PE must use the "Get Return/Status" IOCTL (refer to the section on "Get Return/Status", in "IOCTL", in this document). The value returned in the buffer by the IOCTL may be one of the following:

0x0000	OK	//	operation successful
0x0081	Message length error	//	invalid message length
0x0082	Invalid message pointer	//	invalid message pointer
0x0083	Out of Memory	//	insufficient memory to complete request
0x00A1	Registered	//	other entity not present
0x00A2	Invalid_Reference	//	invalid Reference_Id
0x00A3	Reference_not_unique	//	re-registered id
0x00A4	Active Register/Bind	//	id already bound
0x00A5	Not Bound	//	invalid unbind request
0x00A6	Missing or Illegal element	//	one or more field(s) are missing or contain invalid element values
0x00A7	No system resources	//	a call-slot is unavailable for this request

5.1.3. AR-UNBIND_Mc request [0x06]

Instructs the ARDD to remove its management plane entry from the appropriate list and notify the entity to which it was bound that it is no longer available. This message should only be sent by the PE.

Byte	Description		
2-3	Bind_Id	//	Bind_Id value returned by the AR-BIND_Mc confirmation
4-5	unassigned		

Return:

Note: If the carry flag is set, an error may have occurred. If Interrupt 21h(33), Function 44h(68), Subfunction 03h(03) ("Send Control Data to a Character Device") was used to send this message, then the value in AX will contain a DOS mapped error value. This value may not represent the ASI error value and should be ignored. To retrieve the actual error/status value the AE or PE must use the "Get Return/Status" IOCTL (refer to the section on "Get Return/Status", in "IOCTL", in this document). The value returned in the buffer by the IOCTL may be one of the following:

0x00	OK	//	operation successful
0x81	Message length error	//	invalid message length
0x82	Invalid message pointer	//	invalid message pointer
0x83	Out of Memory	//	insufficient memory to complete request
0xA2	Invalid_Reference	//	invalid Bind_Id
0xA5	Not Bound	//	invalid unbind request
0xA6	Missing or Illegal element	//	one or more field(s) are missing or contain invalid element values

5.1.4. AR-UNBIND_Up request [0x08]

Instructs the ARDD to remove its user plane entries from the appropriate list and notify the entity to which it was bound that it is no longer available.

Byte	Description		
2-5	Reference_Id	//	set=AEI/PEI pair

Note: The Reference_Id contains a copy of the four octets (AEI/PEI), 2 through 5, as defined in Part 1, Commands chapter, Message Format section.

Return:

Note: If the carry flag is set, an error may have occurred. If Interrupt 21h(33), Function 44h(68), Subfunction 03h(03) ("Send Control Data to a Character Device") was used to send this message, then the value in AX will contain a DOS mapped error value. This value may not represent the ASI error value and should be ignored. To retrieve the actual error/status value the AE or PE must use the "Get Return/Status" IOCTL (refer to the section, IOCTL; Get Return/Status, in this document). The value returned in the buffer by the IOCTL may be one of the following:

0x0000	OK	//	operation successful
0x0081	Message length error	//	invalid message length
0x0082	Invalid message pointer	//	invalid message pointer
0x0083	Out of Memory	//	insufficient memory to complete request
0x00A2	Invalid_Reference	//	invalid Reference_Id
0x00A5	Not Bound	//	invalid unbind request
0x00A6	Missing or Illegal element	//	one or more field(s) are missing or contain invalid element values

5.2. ARDD Confirmation Messages

"Confirmation" messages are sent in response to a "request". This message is sent by the ARDD to the AE or PE which sent the "request" using the AR_Callback method described during the bind procedure.

5.2.1. AR-BIND_Mc confirmation [0x01]

Provides the called entity with an acknowledgment that its management plane has been bound with the entity for which it registered.

Byte	Description		
2-3	Bind_Id	//	the id of the bound AE
4-5	unassigned		
6-9	Mc_Callback	//	far address of M/C callback slot (offset, segment)

Note: Refer to the AR_BIND_Mc request for an explanation of the Bind_Id field.

Note: The Bind_Id has no meaning to an AE which receives this message. An AE should ignore this field.

Return:

0x0000	OK	//	operation successful

5.2.2. AR-BIND_Up confirmation [0x03]

Provides the called entity with an acknowledgment that its user plane has been bound with the entity for which it registered.

Byte	Description		
2-5	Reference_Id	//	set=AEI/PEI pair
6-9	Up_Callback	//	far address of user plane callback slot (offset, segment)

Note: The Reference_Id contains a copy of the four octets (AEI/PEI), 2 through 5, as defined in Part 1, Commands chapter, Message Format section.

Return:

0x0000	OK	//	operation successful

5.3. ARDD Indication Messages

"Indication" messages are sent to the AE or PE by the ARDD, using the AR_Callback method described during the bind procedure. These messages are used to inform the AE or PE of an unsolicited event.

5.3.1. AR-UNBIND_Mc indication [0x05]

Provides the called entity with an indication that the entity with which its management plane was bound is no longer available.

Byte	Description		
2-3	Bind_Id	//	Bind_Id value returned by the AR-BIND_Mc confirmation
4-5	unassigned		

Return:

0x0000	OK	//	operation successful

5.3.2. AR-UNBIND_Up indication [0x07]

Provides the called entity with an indication that the entity with which its user plane was bound is no longer available.

Byte	Description		
2-5	Reference_Id	//	set=AEI/PEI pair

Note: The Reference_Id contains a copy of the four octets (AEI/PEI), 2 through 5, as defined in Part 1, Commands chapter, Message Format section.

Return:

0x0000	OK	//	operation successful

5.3.3. AR-UNBIND_ALL indication [0x09]

Provides the called entity with an indication that the entity, with which its management and/or user planes were bound, is no longer available.

Byte	Description		
2-3	Bind_Id	//	the id of the bound AE
4-5	unassigned		

Note: The Bind_Id has no meaning to an AE which receives this message. An AE should ignore this field.

Return:

0x0000	OK	//	operation successful

5.3.4. AR-LOST_MESSAGE indication [0x10]

Provides the called entity with an indication that the one or more messages were lost while it was inactive.

There is no further information supplied with this message.

Return:

0x0000	OK	//	operation successful

5.4. ARDD Response Messages

"Response" messages are sent in response to an "Indication". These messages are sent by the AE or PE to the ARDD using one of the supported transfer mechanisms.

Note: There are no "response" primitives in the "Enhanced DOS/Protected Mode Shell Access Method". The explanation of the primitive was listed as a general definition for the ASI. Future enhancements to this document may contain "response" primitives.

6.0. IOCTL Facilities

This section provides detailed information on the use of the “General IOCTL” facilities supported by the ARDD. The Major code for all General IOCTLs must be 0x0A.

6.1. Get Device Information [Minor Code=80h]

The “Get Device Information” IOCTL may be used by the AE or PE to acquire information on the support level and parameters of a particular implementation of the ARDD. The information shall be placed in a buffer passed to the ARDD during the General IOCTL call.

On entry the first word in the buffer should be set to the length of the buffer receiving the information. The ARDD will place as many bytes of information as it can, into the buffer, given the size of the buffer. The ARDD will not provide partial information. A returned structure element will not be truncated.

On Exit the buffer will have been populated by the ARDD with as much information as it can fit into the buffer given.

This includes:

- ARDD major and minor version number.
- Vendor string.
- Vendor major and minor version number.
- AE device count (bound and unbound).
- UP callback slots.

Note: Refer to Appendix L; Get Device Information Structure for the byte order of the returned buffer.

6.2. Get Return/Status [Minor Code=81h]

The “Get Return/Status” IOCTL may be used by the AE or PE to get the return/status value of the last ARDD command request. The buffer supplied by the caller of this facility will be populated with the last value returned in the status field of the DOS device driver request packet. The buffer should be large enough to hold a 2 byte (1 word) return code. The value returned shall be comprised of a zero byte followed by the value of the lower 8 bits of the status field.

6.3. Get Header Address [Minor Code=82h]

The “Get Header Address” IOCTL may be used to gain access to the ARDD using direct calls to the ARDD's “strategy” and “interrupt” routines. These routines are defined by DOS and are used to pass request packets to a device driver. The buffer returned by this facility will be populated with the far address of the ARDD header. The buffer should be large enough to hold a segment and offset value (4 bytes).

Note: Returned value is comprised of an offset in bytes 1-2 followed by a segment address in bytes 3-4.

Once an entity has acquired the ARDD header address it may format a device driver packet (refer to the section titled “Building a Device Driver Request Packet”, in this document), and pass it to the ARDD in the same manner as DOS would. That is loading ES:BX with the location of the request packet, calling the ARDD's “strategy” routine then calling the ARDD's “interrupt” routine. Upon return the lower byte of the status word will contain an ASI defined return/status value.

Note: The segment address of the “strategy” and “interrupt” routines are the same as the segment address of the device header.

7.0. INT 2Fh Facilities

This section details INT 2Fh facilities used by the ARDD and PM shells.

This includes service used by:

- ARDD (For AE entity access)
- ARDD (For PE entity access)
- Microsoft Windows
- MS-DOS Task Switcher (DOSSHELL.EXE)
- Other Protected Mode Shells (for further study)

7.1. ARDD

The following sections detail INT 2Fh facilities supported by an ARDD implementation.

7.2. ARDD INT 2Fh Facilities - AE Access

This section details INT 2Fh facilities used by the ARDD for inter-device communications between an AE and the ARDD. The AH register should be loaded with the value passed on the command line to the ARDD (AEM) (refer to item 6 of the section titled “ARDD Specifications”, in this document, for further details)

7.2.1. “Query Support” [AL=00h]

This is a DOS defined command passed to the INT 2Fh chain to query whether a INT 2Fh handler has been installed for this id. The ARDD handler routine should return with AL=0FFh.

7.2.2. “Open” [AL=10h]

This command will open the ARDD for access. This call is intended for use by other device drivers (i.e., an AE implemented as a device). If this call is successful, the ARDD should block any subsequent DOS open requests.

On entry:

DS:SI = address of buffer containing a valid, ASI filename, formatted as a NULL terminated ASCII string.

Return:

If carry flag clear:

AX=handle of open device (ARDD generated)

If carry flag set, AX may contain one of the following errors:

0002h	file not found
0005h	access denied

7.2.3. “Close” [AL=20h]

Closes the ARDD and makes it available for access by other entities. Once closed the ARDD may permit DOS open requests.

On entry:

BX = file handle to close

Return:

If carry flag clear:

function completed successfully

If carry flag set, AX may contain the following error:

0006h	invalid file handle

7.2.4. “IOCTL” [AL=30h]

The caller may use this to send requests to the ARDD.

On entry:

BX = file handle
DS:SI = address of message buffer
CX = number of bytes in buffer

Return:

If carry flag clear:

AX=number of bytes accepted by ARDD

If carry flag set, AX may contain the following error:

0005h	access denied
0006h	invalid file handle
000dh	invalid data

7.2.5. “GENIOCTL” [AL=40h]

The caller may use this to send command/requests to the ARDD.

On entry:

BX = file handle
DS:SI = address of message buffer
CH = major code
CL = minor code

Return:

If carry flag clear:

function completed successfully

If carry flag set, AX may contain the following error:

0005h	access denied
0006h	invalid file handle
000dh	invalid data

7.3. ARDD INT 2Fh Facilities - PE Access

This section details INT 2Fh facilities used by a VxD to pass information about various VM activities to the ARDD. This facility is most commonly used to inform the ARDD about the termination of a PE. This will allow the ARDD to post AR-UNBIND_ALL indications to any AEs with which the PE had a connection. The AH register should be loaded with the value passed on the command line to the ARDD (PEM) (refer to item 1 of the section titled “Microsoft Windows Enhanced Mode Support” in “ARDD Specifications”, in this document).

7.3.1. “Query Support” [AL=00h]

This is a DOS-defined command passed to the INT 2Fh chain to query whether a INT 2Fh handler has been installed for this id. The ARDD handler routine should return with AL=0FFh.

7.3.2. “BROADCAST” [AL=10h]

This will be used by a VxD to send general broadcast messages to the ARDD pertaining to the destruction of VMs.

On entry:

BX = VMID
CX = VM state (see table below)

VM state may be one of the following:

VM_SUSPEND	10h
VM_RESUME	20h
VM_NOT_EXECUTEABLE	30h

Return:

A broadcast may not be returned in error.

7.3.3. “DEBUG” [AX=0FFFFh]

If the VxD is built with debugging enabled, then a broadcast will be sent with AX=0FFFFH during the real mode initialization routine.

On entry:

Registers set with values passed the Real-Mode-Init routine by Microsoft Windows (refer to the “Microsoft Windows Device Development Kit” for more information).

BX = flags	bit 0: device previously loaded (in “system.ini”) bit 1: duplicate device id from INT 2Fh device list bit 2: device loaded by an INT 2Fh service

EDX = value placed in the reference DWORD of the Startup Structure (PEM).

SI = environment segment passed to DOS loader

DI = VMM version number (passed to init code in AX)

CS = DS = ES = segment of loaded code and data.

Return:

A broadcast may not be returned in error.

7.4. Microsoft Windows Enhanced Mode Shell

This section details INT 2Fh facilities used by Microsoft Windows.

Note: Refer to the “Microsoft Windows Device Development Kit Virtual Device Adaptation Guide Appendix D” for further information on these and other INT 2Fh facilities made available by Microsoft Windows for use by real mode device drivers.

7.4.1. Call-In Service

A call-in service is a service available to the ARDD to communicate with the Microsoft Windows Enhanced Kernel.

7.4.1.1. Enhanced Windows Installation Check [AX=1600h]

<FOR FURTHER STUDY>

7.4.1.2. Release Current VM's Timeslice [AX=1680h]

<FOR FURTHER STUDY>

7.4.1.3. Begin Critical Section [AX=1681h]

<FOR FURTHER STUDY>

7.4.1.4. End Critical Section [AX=1682h]

<FOR FURTHER STUDY>

7.4.1.5. Get Current VMID [AX=1683h]

<FOR FURTHER STUDY>

7.4.1.6. Switch VMs and Callback [AX=1685h]

<FOR FURTHER STUDY>

7.4.2. Call-Out Service

A call-out service is a service which the Microsoft Windows Enhanced Kernel uses to pass information to the ARDD.

7.4.2.1. Enhance Windows and 286 Extender Init [AX=1605h]

<FOR FURTHER STUDY>

7.4.2.2. Enhance Windows and 286 Extender Exit [AX=1606h]

<FOR FURTHER STUDY>

7.4.2.3. Enhance Windows Init Complete [AX=1608h]

<FOR FURTHER STUDY>

7.4.2.4. Enhance Windows Begin Exit [AX=1609h]

<FOR FURTHER STUDY>

7.5. Microsoft Windows Standard Mode Shell

This section details INT 2Fh facilities used by Microsoft Windows when running in standard mode.

<FOR FURTHER STUDY>

7.6. MS-DOS Task Switcher (DOSSHELL.EXE)

This section details INT 2Fh facilities used by the MS-DOS Task Switcher.

<FOR FURTHER STUDY>

8.0. Callback Function Definitions

This ASI access method defines three types of callback routines:

ARMF	**ARDD Management Function**
M/C	**Management/Control plane**
UP	**User Plane**

These callback routines provide for the asynchronous transfer across the interface in one direction. The exception is the ARDD's ARMF callback which is used by the ARDD to provide indications and confirmations to both the AE and PE.

This section provides the calling sequences for the various callback functions. All of the callbacks provide the same return codes as defined in Appendix A.

If successful, the carry flag will be cleared and the AX register will be set to 0. If the transfer fails, the carry flag will be set and AX will contain an error code (refer to Appendix A).

Note: The callback procedure is responsible for preserving all registers other than AX.

Note: The caller is responsible for maintaining stack integrity.

8.1. AR Management Function Callback Definition

The AR Management Function (ARMF) callback transfers the thread of execution and the message from the ARDD to the called entity. The following calling procedure will be used:

```
PUSH        WORD     MsgSize            ; Bytes in message
PUSH        WORD     seg MsgPtr         ; Segment of message (ARDD formatted buffer --  see
                                          the section, ARDD Command Set Description,
                                          section, in this document)
PUSH        WORD     offset MsgPtr      ; Offset of message.

CALL        AR_Callback
```

8.2. Management/Control Plane Callback Definition

The Management/Control Plane callback transfers the thread of execution and the message from the calling entity to the called entity (i.e., PE -> AE or AE -> PE). The following calling procedure will be used.

```
PUSH        WORD     MsgSize            ; Bytes in message
PUSH        WORD     seg MsgPtr         ; Segment message (ASI formatted buffer --  see
                                          Command section in Part 1)
PUSH        WORD     offset MsgPtr      ; Offset of message

CALL        Mc_Callback
```

8.3. User Plane Callback Definition

The User Plane callback transfers the thread of execution and the message from the calling entity to the called entity (i.e., PE -> AE or AE -> PE). The following calling procedure will be used.

```
PUSH        WORD     DataSize           ; Bytes in data buffer
PUSH        WORD     seg DataPtr        ; Segment of data buffer (protocol specific)
PUSH        WORD     offset DataPtr     ; Offset of data buffer (protocol specific)
PUSH        WORD     CtlSize            ; Bytes in control buffer
PUSH        WORD     seg CtlPtr         ; Segment of control buffer (protocol specific)
PUSH        WORD     offset CtlPtr      ; Offset of control buffer (protocol specific)

CALL        Up_Callback
```

Note: The Control Message may not be required for most data transfers. The Control Message should be used when a protocol requires the passing of control information to upper layers which are synchronized with a data message. When a control message is not present, the CtlSize should be set to zero and CtlPtr set to NULL (0). CtlSize being set to zero will inform the receiving entity to ignore CtlPtr. When a data message is not present, the DataSize should be set to zero and DataPtr set to NULL (0). DataSize being set to zero will inform the receiving entity to ignore DataPtr.

9.0. Registration and Binding Process

To provide compatibility with the rest of the ASI, the ARDD binding procedure is asynchronous. In order to provide this asynchronous operation, the request and confirm operations are non-symmetrical. The request operation is carried out using one of the transfer mechanisms supported in this specification.

These include:

- Interrupt 21h(33), Function 44h(68), Subfunction 03h(03) ("Send Control Data to a Character Device")
- Far calls to the ARDD's "strategy" and "interrupt" routines
- Multiplex Interrupt (INT 2Fh)

The calling program must open the device driver prior to performing any function calls. The confirm operation is carried out when the ARDD posts a bind confirmation to the AR Management Function (ARMF) of the appropriate entity using the method of access supplied during the bind request.

9.1. Registration and Binding Using IOCTLs

The ARDD is a two-sided device (AE and PE entry points). This allows the ARDD to easily differentiate between entity's types which need to be bound since each entity will always call upon a specific entry point. When a bind request arrives at each entry point carrying the same ID, the ARDD notifies both entities of the resulting match. Also, by having this two -sided mechanism which always remains open (while the entity is active), the ARDD is capable of detecting when a process goes away prematurely and can send AR-UNBIND_ALL indications to all entities which have an active bind with the entity that went away. This provides for graceful cleanups.

The following scenarios represent typical load and address resolution procedures. It is assumed that all entities are well-behaved relative to their environment and follow proper procedural practices. This scenario makes the assumption that the AE was loaded first. With this assumption, the sequence of these events must be observed. These scenarios also assume no errors, either logical or physical, occur.

While this scenario is typical, it in no way implies that it is the only valid sequence which can be used.

1. The Address Resolution Device Driver (ARDD) loads and initializes.
2. The ASI Entity (AE) loads. It then opens the ARDD using the name "ARDDAEnn". Using the device handle returned on the open, the AE formats an AR-BIND_Mc request and issues it to the ARDD using one of the supported transfer mechanisms (refer to the section titled "Basic ARDD Specifications", in this document for further details) (Bind_Id=0x00000000). This request contains the AE's M/C plane function address and the AE's Address Resolution Management Function (ARMF) address.

   Note: Do not close the ARDD.

3. The ARDD registers the M/C and ARMF callback routines and returns a value of 0xA1 (Registered) in the status field of the device driver request packet. The error (bit 15) and done (bit 8) are also set.

Note: If Interrupt 21h(33), Function 44h(68), Subfunction 03h(03) ("Send Control Data to a Character Device"), was used to send the AR-BIND_Mc request to the ARDD, then the AE will have the carry flag set. The AX register will contain a DOS mapped error value. This value may not represent the ASI error value and should be ignored. To acquire the ASI error value the AE must place a "General IOCTL" (Interrupt 21h(33), Function 44h(68), Subfunction 0Ch(12)) call with a category of 0Ah(10) (ARDD), minor code of 81h(129) ("Get ARDD Status/Return"), and a pointer to a word to receive the actual status/error. The value stored in this location should be 0x00A1 (Registered).

4. The Program Entity loads. It then opens the ARDD using the name "ARDDPE00". Using the device handle returned on the open, the program entity formats an AR-BIND_Mc request and issues it to the ARDD using one of the supported transfer mechanisms (refer to the section, Basic ARDD Specifications, in this document) (Bind_Id=0x00000000). This request contains the PE's M/C plane function address and the PE's Address Resolution Management Function (ARMF) address.

Note: If Interrupt 21h(33), Function 44h(68), Subfunction 03h(03) ("Send Control Data to a Character Device") was used to send the AR-BIND_Mc request to the ARDD, the PE will have the carry flag set. The AX register will contain a DOS mapped error value. This value may not represent the ASI error value and should be ignored. To acquire the ASI error value the AE must place a "General IOCTL" call (Interrupt 21h(33), Function 44h(68), Subfunction 0Ch(12)), with a category of 0Ah(10) (ARDD), minor code of 0x81 ("Get ARDD Status/Return"), and a pointer to a word to receive the actual status/error. The value stored in this location should be 0x00A1 (Registered).

Note: Steps 5 through 8 occur while the ARDD is servicing the interrupt routine which resulted from the IOCTL call in step 4.

5. The ARDD registers the M/C and ARMF addresses of the PE.
6. The ARDD calls the PE's ARMF with an AR-BIND_Mc confirmation, passing the AE's M/C function address. The callback returns 0 to the ARDD.
7. The ARDD calls the AE's ARMF with an AR-BIND_Mc confirmation, passing the PE's M/C function address. The callback returns 0 to the ARDD.
8. The ARDD returns 0 to the PE (this completes the call sequence from step 4).

Note: At this point the PE and AE are bound. The PE can initiate a call to the AE's management/ control plane function. Via this function, configuration of the ASI Entity can take place (see Figure 3 in this Document).

9. The PE calls the ARDD with an AR-BIND_Up request, passing the user plane and control function (far) addresses (Reference_Id=0x00000005). The ARDD returns 0xA1 to the PE.
10. The Program Entity calls the AE control plane function with an Nb-CONNECT request (PEI=0x0005).
11. The AE calls the ARDD with an AR-BIND_Up request, passing the user plane, and control function (far) addresses (Reference_Id=0x00060005).
12. The ARDD calls the PE with an AR-BIND_Up confirmation, passing the AE's user plane function (far) address (Reference_Id=0x00060005).
13. The ARDD calls the AE with an AR-BIND_Up confirmation, passing the PE user plane function (far) address (Reference_Id=0x00060005).
14. Here, we assume that the appropriate processing goes on in the AE, and an ISDN SETUP is sent to the network.
15. The ARDD returns 0x00 to the AE.

16. A CONNECT is received from the network, which then generates an interrupt to transfer the thread of execution to the AE's interrupt handler.
17. The AE calls the PE control plane function with an Nb-CONNECT confirmation (AEI=0x0006). The PE returns 0x00.
18. The AE issues an interrupt return (IRET) and transfers the thread of execution back to the PE.

Now each of the PE and AE can effect data transfers by calling the other's data and control functions.

The user plane callback function address provides a mechanism for passing data-synchronous user plane control information. The use of this information will be specific to the peer-to-peer data protocol employed across the ISDN connection. The format of the user plane control messages are session-dependent as defined in section 6 of part 1.

Note: The PE's callback routine should only queue messages and return immediately so that the thread of execution can travel back and forth effectively. Once the AE returns control to the PE, the PE can then check the messages which have been posted, and act on them accordingly.

9.2. Registration and Binding Using INT 2Fh Facilities

<FOR FURTHER STUDY>

9.3. Registration and Binding Using Direct ARDD Calls

<FOR FURTHER STUDY>

10.0. Making Device Driver Calls Directly

This section details procedures to follow in order to create a device driver request packet. This packet may be sent directly to the ARDD without utilizing any DOS facilities.

Note: The AE or PE must first open the ARDD before using these facilities.

10.1. Acquiring the Device Header Address

<FOR FURTHER STUDY>

10.2. Building a Device Driver Request Packet

This section details the procedures used by an AE or PE to send device driver request packets directly to the ARDD. The request packet is comprised of two components: a request header which is a fixed length and a command specific portion which has a length dependent upon the request command.

To pass a device driver request packet the AE or PE must format a packet in the same manner as DOS does (refer to the “Microsoft's MS-DOS Programmer's Reference” (Document Number SY0766b-R50-0691)). Once that packet has been prepared the entity must follow the procedures in the order below.

1. Place a far call to the ARDD's strategy routine with the address of the request header in ES:BX.
2. Place a far call to the ARDD's interrupt routine.

On return from the call the lower byte of the Status portion of the request header will contain the ASI defined return/ status value from the ARDD.

10.2.1. Request Header

```
DB      ?                   ; Length in bytes of request header
DB      ?                   ; Not used for character devices
DB      ?                   ; Command code field
```

```
DW       ?                    ; Status
DB       8 dup (?)            ; Reserved
```

Upon entry, the "Status" field contains the following:

Bit 15	;	Error bit
Bit 14-10	;	Reserved
Bit 9	;	Busy
Bit 8	;	Done
Bit 7-0	;	ASI defined Error code (bit 15 on)

Bit 15, the error bit, is set by the device driver if an error is detected or if an invalid request is made to the driver. The low 8 bits indicate the error code.

Bit 9, the busy bit, is not used by the ARDD in this specification.

Bit 8, the done bit, is set by the device driver when the operation is complete.

Error codes are listed in Appendix A.

10.2.2. Device Driver Command Codes

The "Command Code" field in the request header must contain a valid device driver command.

The following values are valid command codes:

0	INIT
1*	MEDIA CHECK (block devices)
2*	BUILD BPB (block devices)
3*	IOCTL INPUT
4*	INPUT (read)
5*	NONDESTRUCTIVE INPUT NO WAIT
6*	INPUT STATUS
7*	INPUT FLUSH
8*	OUTPUT (write)
9*	OUTPUT WITH VERIFY
10*	OUTPUT STATUS
11*	OUTPUT FLUSH
12	IOCTL OUTPUT
13	DEVICE OPEN
14	DEVICE CLOSE
15*	REMOVABLE MEDIA (block devices)
16*	OUTPUT UNTIL BUSY
17-18*	Not defined
19	GENERIC IOCTL
20-22*	Not defined
23*	GET LOGICAL DEVICE
24*	SET LOGICAL DEVICE

Unsupported or illegal commands are marked with an "*". If any of these commands are presented to the ARDD, it will set the error bit and return the with an error value of Unknown Command (03h).

10.2.2.1. INIT Command [00h]

```
DB       13 dup (0)           ; Request header
DB       ?                    ; Not used by character devices
DD       ?                    ; INPUT: end available driver memory
                              ; OUTPUT: end resident code
DD       ?                    ; INPUT: addr CONFIG.SYS device= line
                              ; OUTPUT: Not used by character devices
DB       ?                    ; INPUT: first drive number
DW       ?                    ; OUTPUT: error-message flag
```

This call is made only once, when the device is installed. The INIT command comes directly from DOS and should not be sent to the ARDD directly. The request packet is defined here for reference only.

10.2.2.2. IOCTL Output Command [0Ch]

```
DB       13 dup (0)          ; Request header
DB       ?                   ; Not used by character devices
DD       ?                   ; INPUT: buffer address
DW       ?                   ; INPUT: number of bytes requested
                             ; OUTPUT: number of bytes written
```

The transfer address points to the message block being send to the ARDD.

10.2.2.3. Open and Close Command [0Dh and 0Eh]

```
DB           13 dup (0)                    ; Request header
```

The “open” and “close” requests require no further information. These calls should NOT be made directly to the ARDD. The “open” command will not be available to an entity since you must have the device open to access the “Get Header Address” IOCTL.

Note: The “close” request must NEVER be sent to the ARDD or further access may not be possible.

10.2.2.4. General IOCTL Command [013h]

```
DB       13 dup (0)          ; Request header
DB       ?                   ; INPUT: device category
DB       ?                   ; INPUT: minor code
DD       ?                   ; reserved
DD       ?                   ; INPUT: IOCTL data address
```

The transfer address points to a control block that is used to communicate with the ARDD. The major and minor function codes determine the request that is being made.

Appendix A. Error codes

A.1. General Error Codes

0x0000	OK	//	operation successful
0x0081	Message length error	//	invalid message length
0x0082	Invalid message pointer	//	invalid message pointer
0x0083	Out of Memory	//	insufficient memory to complete request
0x0084	Function not supported	//	requested function not supported
0x0085	Interface Busy	//	message interface busy, message rejected
0x0086	Interface unavailable	//	the PE is temporarily unavailable
0x0087	Lost message	//	one or more messages have been lost

A.2. Bind Procedure Error Codes

0x00A1	Registered	//	other entity not present
0x00A2	Invalid_Reference	//	invalid Reference_Id
0x00A3	Reference_not_unique	//	re-registered id
0x00A4	Active Register/Bind	//	id already bound
0x00A5	Not Bound	//	invalid unbind request
0x00A6	Missing or Illegal element	//	one or more fields are missing or contain invalid element values
0x00A7	No system resources	//	a call-slot is unavailable for this request
0x00FF	Unknown error	//	General failure, no further information is available

Appendix B. Future Enhancements

Future enhancements to this document include:

- Microsoft Windows GUI Library
- Microsoft Windows Protected Mode
- MS-DOS Task Switcher (DOSSHELL.EXE) support
- Buffer Management Protocol
- Other Protected Mode Shells are for further study

Appendix C. Glossary of Terms

The following terms and definitions are used in this document:

AE	The Application Entity refers to the portion of the ASI supplied by the ISDN vendor.
AEM	The Application Entity Multiplex ID is used by the AE device driver to communicate with the ARDD.
AH	The upper 8 bits of the AX register.
AL	The lower 8 bits of the AX register.
API	Application Program Interface. A piece of software that extends access to facilities through a standard library of function calls.
AR	The Address Resolution is a logical software plane that is used to satisfy binding requests.
ARDD	The Address Resolution Device Driver is a device driver defined in this document. Used to manage communications between the AE and PE.
ASI	Applications Software Interface. An open standard API for ISDN. See Part 1.
AX	A general-purpose data register.
BP	A base pointer register.
BX	A general-purpose register.
Byte	A basic unit of data storage and manipulation. A byte is equivalent to 8 bits.
Call	An assembly language instruction telling the processor to execute a subroutine.
Carry flag	The bit in the flag register that indicates whether the previous operation resulted in a carry out or borrow into the high-order bit of the resulting byte of word. Also used by DOS to indicate occurrence of an error.
CS	The code segment register.
CX	A general-purpose register usually used for counting functions.
DI	The destination index register. Paired with ES (ES:DI) to form a far address to a data buffer. Primarily used in transfer instructions as a far pointer to the destination buffer.
DLL	Dynamic Link Library. Commonly used by operating systems to allow for common program functions to be shared among many applications.
DOS	Disk Operating System.
DPMI	DOS Protected Mode Interface. This is the standard interface designed by Microsoft, Intel and IBM to allow an application running in a DOS machine to have access to facilities and memory available to a process while operating in the protected mode,
DS	The data segment register.
DX	A general-purpose register.
Entry point	A character device within the ARDD that is used to provide service to an AE or PE.
EQU	An assembly language directive (equate).
ES	The extra segment register.
FAR	A far piece of code or data is considered to be within a different segment of memory than the current segment. Addressing such addresses requires both segment and offset pointers.
Function	A self-contained coding segment designed to do a specific task. A function is sometimes referred to as a procedure or subroutine.

Hexadecimal	A numbering system based on 16 elements. Digits are numbered 0 through F, as follows: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.
IOCTL	Input/Output Control commands are used to provide control information to devices from a higher level of software.
ISDN	Integrated Services Digital Network, a high-speed digital facility provided by the telephone network.
Linking	The process of resolving external references and address references.
M/C	The Management/Control plane is a logical software plane used to pass management and command.
Nibble	A nibble is 1/2 of a byte. This usually represents 4 bits.
Offset	A distance from a given paragraph boundary in memory. The offset is usually given as a number of bytes.
PE	The Program Entity is the program that accesses the ASI at reference point B.
PEM	The Program Entity Multiplex ID is used by a VxD to communicate information to the ARDD's PE entry point.
PM	See Protected Mode.
Protected Mode	An alternative operating mode of the 80286 and above processors. Protected mode causes a different use of the segment registers and memory than real mode.
Register	The data-holding area, usually 16 bits in length, used by the processor in performing operations.
Segment	A particular area of memory, 64K in size (1K=1024).
Segment register	Any of the CPU registers designed to contain a segment address. They include the CS, DS, ES and SS registers.
SI	The source index register. Paired with DS (DS:DI) to form a far address to a data buffer. Primarily used in transfer instructions as a far pointer to the source buffer.
SP	The stack pointer register.
SS	The stack segment register.
Stack	An area of memory set aside for temporary storage of values in a computing environment. The stack operates in a LIFO (Last In First Out) fashion.
TSR	Terminate but Stay Resident. This type of application may return processing to DOS via a special DOS function while remaining resident in memory.
VM	A Virtual Machine is a mode that processors, of the class 80386 and higher, may enter to allow applications written for DOS to function within a protected mode environment.
VMID	This is a value assigned by the VMM in the operating kernel to reference an instance of a VM. Each running VM is assigned a unique VMID.
VMM	The Virtual Machine Manager is the portion of the kernel that is responsible for the management of VMs.
VxD	A Virtual Device driver is used by processors, of the 80386 and higher class, to virtualize service of a real mode device to a VM. This allows management of system resources in an environment where many applications are running simultaneously.

Appendix D. Sample Code -- DOS Functions

The following sections provide examples of the DOS functions used to communicate with the ARDD. For further information, see Microsoft's MS-DOS Programmer's Reference (Document Number SY0766b-R50-0691).

D.1. Open File with Handle (0x3D)

To open the ARDD, place the device name “ARDDAEnn” or “ARDDPE00” in a buffer and call DOS with a pointer to that buffer.

Example:

```
             mov      dx, seg FileName     ; get segment of buffer
             mov      ds, dx               ;   and place it in ds
             mov      dx, offset FileName  ; ds:dx points to device name
             mov      al, 12h              ; OPEN_ACCESS_READWRITE | OPEN_SHARE_DENYREADWRITE
             mov      ah, 3Dh              ; “Open File with Handle”
             int      21h                  ; call DOS for service
             jc       error_handler        ; if carry set, we had an error
             mov      Handle, ax           ; store handle for later
             .
             .
             .

FileName     DB       'ARDDAE00',0         ; file name buffer
Handle       DW       ?                    ; handle returned from open
```

Return:

If carry flag clear:

AX=Handle of open device.

If carry flag set, AX may contain one of the following errors:

0002h	File not found
0003h	Path not found
0004h	Too many open files
0005h	Access Denied
000Ch	Invalid Access

D.2. Close File with Handle (0x3E)

To close the ARDD, provide the “Handle” which was returned from the open.

Example:

```
             mov      bx, Handle           ; handle of device
             mov      ah, 3Eh              ; “Close File with Handle”
             int      21h                  ; call DOS for service
             jc       error_handler        ; if carry set, we had an error
             .
             .
             .
             .
Handle       DW       ?                    ; handle returned from open
```

Return:

If carry flag clear:

function completed successfully

If carry flag set, AX may contain one of the following errors:

0006h	Invalid Handle

D.3. Send Control Data to a Character Device (0x4403)

To send commands to the ARDD, use the "Handle" obtained from the open.

Example:

Assume that "Buffer" is a buffer containing a properly formatted ARDD message. "Handle" is a word which received the result of the open. "MaxBytes" is a word which contains the actual number of bytes in "Buffer". "ActualBytes" is a word which receives the results of the IOCTL.

```
             mov      bx, Handle          ;  handle of device
             mov      cx, MaxBytes        ; # of bytes to send
             mov      dx, seg Buffer      ; get segment of buffer
             mov      ds, dx
             mov      dx, offset Buffer   ; ds:dx points to data
             mov      ax 4403h            ; "Send Control Data to a Character Device"
             int      21h                 ; call DOS for service
             jc       error_handler       ; if carry set, we had an error
             mov      ActualBytes, ax     ; number of bytes sent
             .
             .
             .
             .
Handle       DW       ?                   ; handle returned from open
ActualBytes  DW       ?                   ; holds number of bytes accepted by ARDD

Buffer       DB       MaxBytes dup (?)    ; the actual number of bytes required by your
                                            application may differ
```

Return:

If carry flag clear:

AX=number of bytes accepted by ARDD

If carry flag set, AX may contain one of the following errors:

0001h	Invalid Function
0005h	Access Denied
0006h	Invalid Handle
000DH	Invalid Data

D.4. General IOCTL (0x440C)

To send a command/request to the ARDD, use the "Handle" obtained from the open.

Example:

Assume that "Buffer" is a buffer containing information to be passed to the ARDD or a buffer to receive information from the ARDD. "Handle" is a word which received the result of the open. "MajorCode" and "MinorCode" represent the command ID for the ARDD function you wish to have performed.

Note: The buffer must be large enough to hold any information returned by the ARDD.

```
             mov      bx, Handle          ; handle of device
             mov      ch, MajorCode       ; major function code
             mov      cl, MinorCode       ; minor function code
             mov      dx, seg Buffer      ; get segment of buffer
             mov      ds, dx
             mov      dx, offset Buffer   ; ds:dx points to data
             mov      ax 440Ch            ; “General IOCTL”
             int      21h                 ; call DOS for service
             jc       error_handler       ; if carry set, we had an error
             .
             .
             .
             .
Handle       DW       ?                   ; handle returned from open
Buffer       DB       200 dup (?)         ; the actual number of bytes required by each call
                                            may differ.
```

Return:

If carry flag clear:

function completed successfully

If carry flag set, AX may contain one of the following errors:

0001h	Invalid Function
0005h	Access Denied
0006h	Invalid Handle
000DH	Invalid Data

D.5. Multiplex Interrupts (INT 2Fh)

To send commands to the ARDD using software interrupt 2Fh follow the procedure below.

Example:

Assume that “Buffer” is a buffer containing information to be passed to the ARDD. “AEM” holds the /AEM value passed the ARDD on the “config.sys” line. “CMD” is one of the command functions as described in section 9 of this document.

Note: The buffer must be large enough to hold any information returned by the ARDD.

```
             mov      ah, AEM             ; /AEM parameter into AH
             mov      al, CMD             ; Command
             mov      cx, MaxBytes        ; # of bytes to send
             mov      dx, seg Buffer      ; get segment of buffer
             mov      ds, dx
             mov      si, offset Buffer   ; ds:si points to data
             .
             .
             load additional parameters
             .
             .
             int      2Fh                 ; call DOS for service
             jc       error_handler       ; if carry set, we had an  error
             .
             .
             .
             .
AEM          DB       ?                   ; /AEM parameter from “config.sys”
Buffer       DB       200 dup (?)         ; the actual number of bytes required by each call
                                            may differ.
```

Return:

Return values and flags are command dependent.

Appendix E. Sample Code -- AE/PE

E.1. General IOCTL Usage

The following sections provide sample code fragments depicting the process used for making general IOCTL calls to the ARDD. These commands are available to either the AE or PE.

E.1.1. Get Device Information [Minor Code=80h]

The following code fragment may be used to query the ARDD for a list of supported facilities.

```
             mov      bx, Handle           ; handle of device
             mov      ch, 0Ah              ; major code for “ARDD”
             mov      cl, 80h              ; minor code = “Get Device Information”
             mov      dx, seg Buffer       ; get segment of buffer
             mov      ds, dx
             mov      dx, offset Buffer    ; ds:dx points to data
             mov      word ptr Buffer, 13h ; number of bytes in buffer
             mov      ax 440Ch             ; “General IOCTL”
             int      21h                  ; call DOS for service
             jc       error_handler        ; if carry set, we had an error
             .
             .
             .
             .
Handle       DW       ?                    ; handle returned from open
Buffer       DB       13h dup (?)          ; buffer to hold ARDD device information
```

Return:

If carry flag clear:

function completed successfully

If carry flag set, AX may contain one of the following errors:

0001h	Invalid Function
0005h	Access Denied
0006h	Invalid Handle
000DH	Invalid Data

E.1.2. Get Return/Status [Minor Code=81h]

The following code fragment may be used to query the ARDD for the value of the last return/status.

```
             mov      bx, Handle           ; handle of device
             mov      ch, 0Ah              ; major code for “ARDD”
             mov      cl, 81h              ; minor code = “Get Return/Status”
             mov      dx, seg Buffer       ; get segment of buffer
             mov      ds, dx
             mov      dx, offset Buffer    ; ds:dx points to data
             mov      ax 440Ch             ; “General IOCTL”
             int      21h                  ; call DOS for service
             jc       error_handler        ; if carry set, we had an error
             .
             .
             .
             .
Handle       DW       ?                    ; handle returned from open
Buffer       DW       ?                    ; buffer to hold return/status from ARDD
```

Return:

If carry flag clear:

function completed successfully

If carry flag set, AX may contain one of the following errors:

0001h	Invalid Function
0005h	Access Denied
0006h	Invalid Handle
000DH	Invalid Data

E.1.3. Get Header Address [Minor Code=82h]

The following code fragment may be used to query the ARDD for a far address to its device header.

```
             mov      bx, Handle          ; handle of device
             mov      ch, 0Ah             ; major code for “ARDD”
             mov      cl, 82h             ; minor code = “Get Header Address”
             mov      dx, seg Buffer      ; get segment of buffer
             mov      ds, dx
             mov      dx, offset Buffer   ; ds:dx points to data
             mov      ax 440Ch            ; “General IOCTL”
             int      21h                 ; call DOS for service
             jc       error_handler       ; if carry set, we had an error
                      .
             .
             .
             .
Handle       DW       ?                   ; handle returned from open
Buffer       DD       ?                   ; buffer to hold segment:offset address of device
                                            header
```

Return:

If carry flag clear:

function completed successfully

If carry flag set, AX may contain one of the following errors:

0001h	Invalid Function
0005h	Access Denied
0006h	Invalid Handle
000DH	Invalid Data

E.2. INT 2Fh Facilities Usage

The following sections provide sample code fragments depicting the process used to communicate to the ARDD using INT 2Fh facilities. These processes may only be used by an AE which has been loaded using an entry in “config.sys”.

E.2.1. INT 2Fh “Open” Function

To open the ARDD, place the device name “ARDDAEnn” in a buffer and call INT 2Fh with a pointer to that buffer.

Example:

```
             mov      ah, AEM             ; /AEM parameter into AH
             mov      al, 10h             ; “open” command
             mov      dx, seg FileName     ; get segment of buffer
             mov      ds, dx               ;   and place it in ds
             mov      si, offset FileName  ; ds:si points to device name
             int      2Fh                  ; call DOS for service
             jc       error_handler        ; if carry set, we had an error
             mov      Handle, ax           ; store handle for later
             .
             .
             .

             FileName DB                   'ARDDAE00',0;file name buffer
Handle       DW       ?                    ; handle returned from open
AEM          DB       ?                    ; holds /AEM parameter
```

Return:

If carry flag clear:

AX=handle of open device.

If carry flag set, AX may contain one of the following errors:

0002h	file not found
0005h	access denied

E.2.2. INT 2Fh "Close" Function

To close the ARDD, provide the "Handle" which was returned from the open.

Example:

```
             mov      ah, AEM              ; /AEM parameter into AH
             mov      al, 20h              ; "close" command
             mov      bx, Handle           ; handle of device
             int      2Fh                  ; call DOS for service
             jc       error_handler        ; if carry set, we had an error
             .
             .
             .
             .
Handle       DW       ?                    ; handle returned from open
AEM          DB       ?                    ; holds /AEM parameter
```

Return:

If carry flag clear:

function completed successfully

If carry flag set, AX may contain the following error:

0006h	invalid file handle

E.2.3. INT 2Fh "IOCTL" Function

To send commands to the ARDD, use the "Handle" obtained from the open.

Example:

Assume that "Buffer" is a buffer containing a properly formatted ARDD message. "Handle" is a word which received the result of the open. "MaxBytes" is a word which contains the actual number of bytes in "Buffer". "ActualBytes" is a word which receives the results of the IOCTL.

```
             mov      ah, AEM              ; /AEM parameter into AH
             mov      al, 30h              ; "IOCTL" command
             mov      bx, Handle           ;  handle of device
             mov      cx, MaxBytes         ; # of bytes to send
             mov      dx, seg Buffer       ; get segment of buffer
             mov      ds, dx
             mov      si, offset Buffer    ; ds:si points to data
             int      2Fh                  ; call DOS for service
             jc       error_handler        ; if carry set, we had an error
             mov      ActualBytes, ax      ; number of bytes sent
             .
             .
             .
             .
Handle       DW       ?                    ; handle returned from open
ActualBytes  DW       ?                    ; holds number of bytes accepted by ARDD
AEM          DB       ?                    ; holds /AEM parameter

Buffer       DB       MaxBytes dup (?)     ; the actual number of bytes required by your
                                             application may differ
```

Return:

If carry flag clear:

AX=number of bytes accepted by ARDD

If carry flag set, AX may contain the following error:

0005h	access denied
0006h	invalid file handle
000dh	invalid data

E.2.4. INT 2Fh "GENIOCTL" Function

To send a command/request to the ARDD, use the "Handle" obtained from the open.

Example:

Assume that "Buffer" is a buffer containing information to be passed to the ARDD or a buffer to receive information from the ARDD. "Handle" is a word which received the result of the open. "MajorCode" and "MinorCode" represent the command ID for the ARDD function you wish to have performed.

Note: The buffer must be large enough to hold any information returned by the ARDD.

```
             mov      ah, AEM              ; /AEM parameter into AH
             mov      al, 40h              ; "GENIOCTL" command
             mov      bx, Handle           ; handle of device
             mov      ch, MajorCode        ; major function code
             mov      cl, MinorCode        ; minor function code
             mov      dx, seg Buffer       ; get segment of buffer
             mov      ds, dx
             mov      si, offset Buffer    ; ds:si points to data
             int      2Fh                  ; call DOS for service
             jc       error_handler        ; if carry set, we had an error
             .
             .
             .
             .
Handle       DW       ?                    ; handle returned from open
AEM          DB       ?                    ; holds /AEM parameter

Buffer       DB       200 dup (?)          ; the actual number of bytes required by each call
                                             may differ.
```

Return:

If carry flag clear:

function completed successfully

If carry flag set, AX may contain the following error:

0005h	access denied
0006h	invalid file handle
000dh	invalid data

Appendix F.

This page intentionally blank

Appendix G.

This page intentionally blank

Appendix H.

This page intentionally blank

Appendix I. Sample Code -- ARDD

This section contains code fragments from a functional ARDD device driver.

I.1. The ARDD Header

The first device driver header is located at offset 0 in the device driver image file. The header defines such things as the device name, location of next device header, if in the same image file, address of the strategy and interrupt routines and an attribute word. The header follows this format:

```
dhLink       DWORD    ?                   ; link to next driver
dhAttributes WORD     ?                   ; device attributes
dhStrategy   WORD     ?                   ; strategy-routine offset
dhInterrupt  WORD     ?                   ; interrupt-routine offset
dhName       BYTE     '????????'          ; device name
```

The dhLink field must be -1 (0FFFFFFFFh) if this is the last device-driver header in the file. Otherwise, the low 16 bits must contain the offset (from the beginning of the load image) to the next device-driver header, and the high 16 bits must contain zero.

The dhAttributes field specifies the device type and provides additional information that DOS uses when creating requests. The bits in this field used for a character device driver, like the ARDD, is as follows:

Bit	Meaning
0	Specifies that the device is the standard input device. This bit must be set to 1 if the driver replaces the resident device driver that supports the standard input device.
1	Specifies that the device is the standard output device. This bit must be set to 1 if the driver replaces the resident device driver that supports the standard output device.
2	Specifies that the device is the NUL device. The resident NUL device driver cannot be replaced. This bit must be zero for all other device drivers.
3	Specifies that the device is the clock device. This bit must be set to 1 if the driver replaces the resident device driver that supports the clock device.
4	For a character-device driver. Specifies that the driver supports fast character output. If this bit is set, DOS issues Interrupt 29h (with the character value in the AL register) when a program writes to the device. During its initialization, the device driver must install a handler (for Interrupt 29h) that carries out the fast output.
6	Specifies whether the device supports the generic IOCTL function. This bit must be set to 1 if the device driver implements Generic IOCTL (Device-Driver Function 13h).
7	Specifies whether the device supports IOCTL queries. This bit must be set to 1 if the device driver implements IOCTL Query (Device-Driver Function 19h).
11	Specifies whether the driver supports Open Device and Close Device (Device-Driver Functions 0Dh and 0Eh). This bit must be set to 1 if the driver implements these functions.
13	Specifies whether the driver supports Output Until Busy (Device-Driver Function 10h). This bit must be set to 1 if the driver implements this function.
14	Specifies whether the driver supports IOCTL Read and IOCTL Write (Device-Driver Functions 03h and 0Ch). This bit must be set to 1 if the driver implements these functions.
15	Specifies whether the driver supports a character device or a block device. This bit must be set to 1 for all ARDD implementations.

Note: Any bits in the dhAttributes field that are not used for a given device type must be zero.

The dhStrategy and dhInterrupt fields contain the offsets to the entry points of the strategy and interrupt routines. Since these fields are 16-bit values, the entry points must be in the same segment as the device-driver file header. For a device driver in a binary image file, the offsets are in bytes from the beginning of the file; for a driver in an .EXE-format file, the offsets are in bytes from the beginning of the file's load image.

The dhName field is an 8-byte field that contains the device name. A character-device driver must supply a logical-device name of no more than eight characters. If it has fewer than eight characters, the name must be left-aligned and any remaining bytes in the field must be filled with space characters (ASCII 20h). The device name must not contain a “:”.

A sample ARDD device header is shown below:

```
org         0

Header1:
            WORD     offset Header2      ; link to next device driver
            WORD     0
            WORD     1100100001000000b   ; device attribute word
            WORD     Strategy1           ; “strategy” routine entry point
            WORD     Interrupt1          ; “interrupt” routine entry point
            BYTE     'ARDDAE00'          ; logical-device name

            .
            .
            .
            .
            somewhere else in the device image file
            .
            .
Header2:
            DWORD    -1                  ; signal end of chain
            WORD     1100100001000000b   ; device attribute word
            WORD     Strategy2           ; “strategy” routine entry point
            WORD     Interrupt2          ; “interrupt” routine entry point
            BYTE     'ARDDPE00'          ; logical-device name
```

I.2. Hooking the INT 2Fh Vector

The following code fragment may be used by the ARDD to hook the INT 2Fh vector. Once hooked, the “NewInt2F” function will be called with every instance of an INT 2Fh.

```
OldInt2F    DWORD    ??                  ; far address of old INT 2Fh vector

ErrorMsg    BYTE     'ARDD unable to be used by Microsoft Windows', 0d, 0a, '$'

NewInt2F    proc     far

            assume   cs:code, ds:nothing, es:nothing

            . code to process INT 2Fh messages
            .
            .
            .
NewInt2Fh   endp

HookInt2f   proc     near

            assume   cs:code, ds:code

            mov      ah, 35h             ; DOS “Get Interrupt Vector” command
            mov      al, 2fh             ;  vector to get (2Fh)
            int      21h

                                         ; returns with es:bx=far address of vector

            mov      ax, es
            mov      word ptr [OldInt2F+2], ax   ; store offset
            mov      word ptr [OldInt2F], bx     ; store segment
```

```
                mov      dx, offset NewInt2F  ; ds:dx=far address of new vector
                mov      ah, 25h              ; DOS “Set Interrupt Vector” command
                mov      al, 2fh              ;  vector to set (2Fh)
                int      21h

                ret

HookInt2f       endp
```

I.3. The INT 2Fh Process

I.3.1. Preventing the loading of Microsoft Windows

An ARDD that has hooked the INT 2Fh vector may use a process similar to the example below to prevent the loading of the Microsoft Windows Standard or Enhanced Mode kernel.

```
OldInt2F        DWORD    ??                   ; far address of old INT 2Fh vector

ErrorMsg        BYTE     'ARDD unable to be used by Microsoft Windows', 0d, 0a, '$'

NewInt2F        proc     far

                assume   cs:code, ds:nothing, es:nothing

                cmp      ax, 1605h            ; is this the Windows startup
                jz       BlockStartup

                jmp      cs:[OldInt2F]        ; call the rest of the chain

BlockStartup:

                push     ds                   ; save ds
                push     cs                   ; Move cs
                pop      ds                   ;  into cs for far pointer to msg

                assume   ds:code

                mov      dx,offset ErrorMsg

                mov      ah,9                 ; display driver sign-on message
                int      21h

                pop      ds                   ; restore original ds

                assume   ds:nothing

                mov      cx,1                 ; block windows start test

                iret

NewInt2F        endp
```

I.4. “Open” Process

<FOR FURTHER STUDY>

I.5. “Close” Process

<FOR FURTHER STUDY>

I.6. “IOCTL” Process

<FOR FURTHER STUDY>

I.7. “GENIOCTL” Process

<FOR FURTHER STUDY>

Appendix J. Sample Code -- VxD

This section contains the source code for a functional Microsoft Windows VxD.

Code was created using the Microsoft Family of programming tools. This includes the Microsoft Windows 3.0 Device Driver Kit with MASM5, LINK386 and EXEHDR.EXE

J.1. ASIARDD.386 Source File (asiardd.asm)

```
             name     ASIARDD
             title    VxD broadcast driver
             page     60,132


             .386p


;********************************************************** ************
;
; Module name: ASIARDD
;
; Description: This is a Windows Enhanced Mode Virtual Device Driver that is
;              responsible for broadcasting VM status messages to an installed
;              ARDD
;
;********************************************************** ************


             .xlist
             include vmm.inc
             .list


ARDD_VM_SUSPEND           EQU   10h
ARDD_VM_RESUME            EQU   20h
ARDD_VM_NOT_EXECUTEABLE   EQU   30h


; Virtual Device declaration

Declare_Virtual_Device ASIARDD, '1', '0', ASIARDD_Control,\
             Undefined_Device_ID, Undefined_Init_Order


; Local Data

VxD_DATA_SEG

Int2fAH      db       0                   ; value to place in AH prior to an INT 2Fh

VxD_DATA_ENDS


VxD_LOCKED_CODE_SEG

;********************************************************** ************
; Control function:
; This is a call-back routine to handle the messages that are sent to VxD's to
; control system operation.
;********************************************************** ************

BeginProc    ASIARDD_Control

             Control_Dispatch Sys_Critical_Init, ProtectedModeInit

             Control_Dispatch VM_Suspend, <short SendSuspend>
             Control_Dispatch VM_Resume, <short SendResume>
             Control_Dispatch VM_Not_Executeable, <short SendNotExecuteable>

             clc
             ret

EndProc      ASIARDD_Control

VxD_LOCKED_CODE_ENDS


VxD_CODE_SEG

BeginProc    SendSuspend

             mov      ebx, [ebx.CB_VMID]

             mov      cx, ARDD_VM_SUSPEND

             mov      ah, cs:Int2fAH
             mov      al, 10h

             push     DWORD PTR 2fh

             VMMcall Exec_VxD_Int

             clc
             ret

EndProc      SendSuspend


BeginProc    SendResume

             mov      ebx, [ebx.CB_VMID]

             mov      cx, ARDD_VM_RESUME

             mov      ah, cs:Int2fAH
             mov      al, 10h

             push     DWORD PTR 2fh

             VMMcall Exec_VxD_Int

             clc
             ret

EndProc      SendResume


BeginProc    SendNotExecuteable

             mov      ebx, [ebx.CB_VMID]

             mov      cx, ARDD_VM_NOT_EXECUTEABLE

             mov      ah, cs:Int2fAH
             mov      al, 10h

             push     DWORD PTR 2fh

             VMMcall Exec_VxD_Int

             clc
             ret

EndProc      SendNotExecuteable


VxD_CODE_ENDS
```

```
VxD_REAL_INIT_SEG


BeginProc     RealModeInitCode

 IFDEF DEBUG

              push      ax
              push      di

              mov       di, ax
              mov       ax, 0ffffh

              int       2fh

              pop       di
              pop       ax

 ENDIF

              cmp       bx, Loading_From_INT2F
              jz        short OkToLoad

              mov       dx, offset ErrVxDLd
              mov       ah, 9
              int       21h

              xor       bx, bx
              xor       si, si
              xor       edx, edx

              mov       ax, Abort_Win386_Load + No_Fail_Message
              ret

ErrVxDLd      db        'Attempt to load multiple instances of ASIARDD.386', 0dh, 0ah, '$'

OkToLoad:

              cmp       edx, 0
              jnz       short IntHandleIsOk

              mov       dx, offset ErrIntHdl
              mov       ah, 9
              int       21h

              xor       bx, bx
              xor       si, si
              xor       edx, edx

              mov       ax, Abort_Win386_Load + No_Fail_Message
              ret

ErrIntHdl     db        'Illegal Int 2Fh value - load aborted', 0dh, 0ah, '$'

IntHandleIsOk:

; remember edx has to be passed to protected mode

              xor       bx, bx
              xor       si, si

              xor       ax, ax

              ret

EndProc       RealModeInitCode


VxD_REAL_INIT_ENDS


VxD_ICODE_SEG
```

```
BeginProc    ProtectedModeInit

             mov      [Int2fAH], dl

             clc
             ret

EndProc      ProtectedModeInit


VxD_ICODE_ENDS

             END      RealModeInitCode      ; real mode entry point
```

J.2. ASIARDD.386 Definition file (asiardd.def)

```
LIBRARY  ASIARDD

DESCRIPTION 'ASIARDD Virtual Broadcast Device'

EXETYPE  DEV386

SEGMENTS
             _LTEXT PRELOAD NONDISCARDABLE
             _LDATA PRELOAD NONDISCARDABLE
             _ITEXT CLASS 'ICODE' DISCARDABLE
             _IDATA CLASS 'ICODE' DISCARDABLE
             _TEXT  CLASS 'PCODE' NONDISCARDABLE
             _DATA  CLASS 'PCODE' NONDISCARDABLE

EXPORTS
             ASIARDD_DDB @1
```

Appendix K. Sample Code -- Protected Mode Access

K.1. DPMI Access Services

<FOR FURTHER STUDY>

This section will contain code fragments demonstrating procedures used to access ASI facilities by DPMI applications.

K.2. Microsoft Windows GUI Applications Access

<FOR FURTHER STUDY>

This section will contain code fragments demonstrates procedures used to access ASI facilities from Microsoft Windows GUI applications.

Appendix L. Structures

This section contains structures used by the ARDD.

L.1. Get Device Information Structure

Byte	Description		
1-2	Length of Buffer	//	Length of returned buffer
3	Major ARDD version number	//	The ARDD major version number
4	Minor ARDD version number	//	The ARDD minor version number
5-12	Vendor string	//	Vendor specific string
13	Vendor major version number	//	Vendor specific major version number
14	Vendor minor version number	//	Vendor specific minor version number
15	Number of AEs	//	Total number of bound AEs
16	Number of AE entry points	//	Number of AE entry points
17	Number of UP callback slots	//	Number of user plane callback slots. This is used by the ARDD during a bind request to provide virtual far call services.

Appendix M. References

M.1 ANS Documents

[1] ANS T1.607-1990, *Telecommunications — Integrated Services Digital Network (ISDN) — Digital Subscriber Signalling System Number 1 (DSS1) — Layer 3 Signalling Specification for Circuit-Switched Bearer Service*.

M.2 CCITT Documents

[2] CCITT Recommendation I.320 - 1988, *ISDN Protocol Reference Model*.

[3] CCITT Recommendation I.515 - 1988, *Parameter Exchange for ISDN Networking*.

[4] CCITT Recommendation Q.921-1988 (also designated CCITT Recommendation I.441-1988), *ISDN User-Network Interface Data Link Layer Specification.*

[5] CCITT Recommendation Q.931-1988 (also designated CCITT Recommendation I.451-1988), *ISDN User-Network Interface — Layer 3 Specification for Basic Call Control*.

[6] CCITT Recommendation V.110 -1988, *Support of Data Terminal Equipments (DTEs) with V-series Type Interfaces by an Integrated Services Digital Network (ISDN).*

[7] CCITT Recommendation V.120 -1988, *Support by an ISDN of Data Terminal Equipment with V-series Type Interfaces with Provision for Statistical Multiplexing.*

[8] CCITT Recommendation X.25 -1984, *Interface between Data Terminal Equipment (DTE) and Data Circuit-Terminating Equipment (DCE) for Terminals Operating in the Packet Mode and Connected to Public Data Networks by Dedicated Circuit.*

M.3 ISO Documents

[9] ISO 8824:1987(E), *Information processing systems — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1).*

[10] ISO 8825:1987(E), *Information processing systems — Open Systems Interconnection — Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).*

M.4 Other Documents

[11] NIST Special Publication 500-183, *Stable Implementation Agreements for Open Systems Interconnection Protocols*, Version 4, Edition 1, December 1990.

[12] *MS-DOS Programmer's Reference* (Document Number SY0766b-R50-0691).